CAPÍTULO 20
INTEGRAÇÃO JDBC COM UMA APLICAÇÃO WEB

A partir desse momento, vamos deixar de usar classes simples com métodos main para realizar os testes de nossos métodos que realizam transações no banco de dados. Nosso projeto web utilizará o JSF2.

O projeto web será composto por telas onde será possível fazer o CRUD de Produtos.

20.1. EXERCÍCIOS: Gravação de dados

1) Abra o Eclipse na área de trabalho.

2) Configure o Tomcat no Eclipse.

3) Crie um novo projeto web:
   a. Acesse o menu **File ➔ New ➔ Project**.
   b. Selecione **Web ➔ Dynamic Web Project** e clique em **Next**.
   c. Em Project Name, coloque *ds1-jdbc-jsf*.
   d. Selecione como *Target Runtime* a versão do Tomcat que acabamos de configurar:
   e. Na seção **Configuration**, clique em **Modify** para acrescentarmos suporte ao JSF.
   f. Na tela que abre, marque a opção **JavaServer faces** e versão **2.0**, depois clique em Ok.
   g. Na tela de criação, clique em **Next** três vezes até chegar à tela de configurações do JSF. Em Type, selecione a opção **Disable Library Configuration**, indicando ao Eclipse que nós mesmos vamos copiar os JARs do JSF.
   h. Ainda nessa tela, em URL Mapping, remova o mapeamento /faces/* e adicione um novo mapeamento como *.xhtml.
   i. Clique em Finish para criar o projeto.

4) Acesse o moodle e localize a pasta JSF-JARS e copie o conteúdo da pasta para dentro da pasta **WEB-INF/lib** do projeto.

5) Acesse a moodle e localize a pasta JDBC e copie o JAR do MySQL para dentro da pasta **WEB-INF/lib** do projeto.

6) Com já fizemos a classe Produto e ProdutoDAO no exercício anterior, vamos vamos importá-las já prontas:
   a. Acesse o moodle e localize a pasta PROJETOS e copie o arquivo *dao-modelo.zip* para seu desktop.
   b. No Eclipse, clique com o botão direito em *ds1-jdbc-jsf*;

c. Escolha a opção **Import Archive File;**

d. Escolha o local desktop e o arquivo *dao-modelo.zip;*

e. Clique em **Finish**.

7) Crie a página de cadastro de produtos (`PRODUTO.XHTML`) no diretório `WEBCONTENT`:

a. Clique com o botão direito em `WEBCONTENT`, acesse a opção **New ➔ Other**

b. Escolha **Web ➔ HTML File** e coloque o nome de *produto.xhtml*

c. Clique em **Next**

d. Selecione a opção **New XHTML File (1.0 transitional)**

e. Clique em **Finish**.

8) Importe as tags core e HTML para sua página. Utilize a tag `<HTML>` de sua página:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
```

9) Substitua as tags `<HEAD>` e `<BODY>` pelas respectivas tags `<H:HEAD>` e `<H:BODY>` do JSF:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitiona
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">

<h:head>
    <meta http-equiv="Content-Type" content="text/html;
    <title>Insert title here</title>
</h:head>

<h:body>

</h:body>

</html>
```

10) Altere o título da página `.XHTML` para "**Sistema de Notas Fiscais**".

11) Acrescente um título HTML na página através da tag `<H:OUTPUTTEXT>`:

```
<h:body>
    <h2>
        <h:outputText value="Novo Produto" />
    </h2>
</h:body>
```

12) Adicione um formulário com a entrada de dados para **nome**, **descrição** e **preço** e um botão para gravar os dados. Utilize o PANELGRID para alinhar os componentes.

```
<h:form>
    <h:panelGrid columns="2">
        <h:outputLabel value="Nome: " for="nome" />
        <h:inputText id="nome"  />
        <h:outputLabel value="Descrição: " />
        <h:inputTextarea id="descricao" for="descricao" />
        <h:outputLabel value="Preço:" for="preco" />
        <h:inputText id="preco" />
        <h:commandButton value="Gravar" />
    </h:panelGrid>
</h:form>
```

13) Crie o Managed Bean para gerenciar e cadastrar os produtos. Crie o pacote *br.com.etec.jdbc.mb* e crie a classe *ProdutoBean* dentro deste. A classe deverá ser anotada com @MANAGEDBEAN do JSF2:

```
@ManagedBean
public class ProdutoBean {

}
```

14) A classe PRODUTOBEAN terá apenas um PRODUTO com seu GETTER e SETTER e um DAO:

```
private Produto produto = new Produto();
ProdutoDAO dao = new ProdutoDAO();

//getter e setter do produto
```

15) Faça o *binding* dos componentes do PRODUTO.XHTML com o atributo PRODUTO que criamos no Managed Bean. Use o atributo **value** dos inputs como a seguir:

```
<h:form>
    <h:panelGrid columns="2">
        <h:outputLabel value="Nome: " for="nome" />
        <h:inputText id="nome" value="#{produtoBean.produto.nome}" />
        <h:outputLabel value="Descrição: " />
        <h:inputTextarea id="descricao" for="descricao"
            value="#{produtoBean.produto.descricao}" />
        <h:outputLabel value="Preço:" for="preco" />
        <h:inputText id="preco" value="#{produtoBean.produto.preco}" />
    </h:panelGrid>
</h:form>
```

16) O próximo é a criação do método que fará a gravação dos dados no banco de dados. Para isso, criaremos o método *grava()* em PRODUTOBEAN:

```
// getter e setter do produto

public void grava() {
    dao.insere(produto);
}
```

17) Ao realizarmos a gravação de um produto, o formulário continuará preenchido. Para limparmos o formulário, basta fazermos o atributo `PRODUTO` apontar para uma instância nova de `PRODUTO`:

```
// getter e setter do produto

public void grava() {
    dao.insere(produto);
    this.produto = new Produto();
}
```

18) Por fim, precisamos fazer o *binding* do botão com o método `GRAVA()` que acabamos de criar. Para isto, altere o código do seu botão no `PRODUTO.XHTML` para adicionar o atributo **action**:

```
<h:commandButton value="Gravar" action="#{produtoBean.grava}" />
```

19) Execute o script a seguir no MySQL:

```
create database jdbc;

use jdbc;

create table produto (
id INT PRIMARY KEY AUTO_INCREMENT,
nome VARCHAR(50),
descricao VARCHAR(255),
preco double);
```

20) Rode a página no servidor e teste a gravação de um produto.

20.2. LISTAGEM COM DATATABLE

Agora que já temos pronto o cadastro de produtos da nossa aplicação, precisamos visualizá-los na nossa tela. Para isso, iremos fazer uma listagem de produtos cadastrados abaixo do formulário.

O componente `<H:DATATABLE>` gera uma tabela HTML que pode ser associada a um Managed Bean para obter dados dinamicamente. Para utilizá-lo, primeiramente temos que indicar de onde os dados virão. Indicamos esses valores no atributo **value** do `DATATABLE` que espera receber uma referência para um método *getter* que retorne um `LIST`, por exemplo.

Dessa forma, para declarar o DATATABLE passaremos via Expression Language uma referência para um getter:

```
<h:dataTable value="#{produtoBean.produtos}">

</h:dataTable>
```

Guardaremos a lista em um atributo do Managed Bean:

```
@ManagedBean
public class ProdutoBean {

    private Produto produto = new Produto();
    private ProdutoDAO dao = new ProdutoDAO();
    private List<Produto> produtos;

    // outros métodos
}
```

A Expression Language #{PRODUTOBEAN.PRODUTOS} está apontando para um método chamado *getProdutos* no Managed Bean PRODUTOBEAN que implementaremos. Ao invocar o método *getProdutos*, devolveremos uma lista de produtos que será buscada no banco de dados.

```
// outros métodos e atributos
public List<Produto> getProdutos() {
    if (produtos == null) {
        produtos = dao.getLista();
    }
    return produtos;
}
```

Para determinarmos como os dados serão exibidos na tela, precisamos alterar um pouco o nosso DATATABLE. O DATATABLE funciona como um laço FOR, no qual você diz qual a COLLECTION será percorrida e qual a variável em que ele armazenará o item atual durante a iteração. No DATATABLE, definimos essa variável temporária através do atributo **var**, no qual podemos dar um nome qualquer, como *produto*, possibilitando o acesso dentro das colunas:

```
<h:dataTable value="#{produtoBean.produtos}" var="produto">

</h:dataTable>
```

No corpo da tag <H:DATATABLE> definiremos as colunas com <H:COLUMN>. Dentro de cada coluna colocamos o valor que queremos renderizar na tela utilizando a Expression Language para recuperar os valores do item a cada iteração:

```
<h:dataTable value="#{produtoBean.produtos}" var="produto">

    <h:column>
        #{produto.nome}
    </h:column>

    <h:column>
        #{produto.descricao}
    </h:column>

    <h:column>
        #{produto.preco}
    </h:column>

</h:dataTable>
```

As tabelas de dados suportam cabeçalhos e rodapés. Os cabeçalhos são úteis para, por exemplo, incluir as descrições das colunas. Os rodapés também são úteis para, por exemplo, exibir cálculos, totalizadores, etc. A inclusão de cabeçalhos e rodapés é conseguido usando a tag `<F:FACET>` com o nome "header" e "footer", para, respectivamente, cabeçalho e rodapé.

```
<h:column>
    <f:facet name="header">Nome</f:facet>
#{produto.nome}
</h:column>
```

Ao acessarmos a página que contém o formulário e gravarmos alguns produtos, a listagem de produtos não é atualizada após a gravação. A questão é que o *binding* do `DATATABLE` é feito no `GETPRODUTOS`, que nada mais faz do que retornar a lista de produtos que, até então, só foi buscada quando uma requisição foi feita para a aplicação. Precisamos recuperar a nova listagem no banco de dados (que agora conterá um produto novo que acabou de ser adicionado) logo após a gravação do mesmo no banco de dados. Para isso, basta adicionar ao fim do método `GRAVA()` uma chamada para o `DAO` que fará essa busca:

```
public void grava() {
    dao.insere(produto);
    this.produto = new Produto();
    this.produtos = dao.getLista();
}
```

20.3. EXERCÍCIOS: Fazendo a listagem de produtos

1) Para listar os produtos cadastrados, adicione à sua classe `PRODUTOBEAN` o atributo *produtos* que é um `LIST` de `PRODUTO` com seu *getter* e *setter*:

```
@ManagedBean
public class ProdutoBean {

    private Produto produto = new Produto();
    private ProdutoDAO dao = new ProdutoDAO();
    private List<Produto> produtos;
    // outros métodos
}
```

2) Faça, no método `GETPRODUTOS()`, a busca no `DAO` e verifique se a busca já não foi feita para evitar várias chamadas ao banco:

```
// outros métodos e atributos
public List<Produto> getProdutos() {
    if (produtos == null) {
        produtos = dao.getLista();
    }
    return produtos;
}
```

3) Adicione abaixo do seu formulário `PRODUTO.XHTML` os componentes para gerar a tabela:

```
<h:dataTable value="#{produtoBean.produtos}" var="produto">

    <h:column>
        <f:facet name="header" >"Nome" </f:facet>
        #{produto.nome}
    </h:column>

    <h:column>
        <f:facet name="header" >"Nome" </f:facet>
        #{produto.descricao}
    </h:column>

    <h:column>
        <f:facet name="header" >"Nome" </f:facet>
        #{produto.preco}
    </h:column>

</h:dataTable>
```

4) Acesse `PRODUTO.XHTML` no seu navegador e veja o resultado final.

5) Adicione produtos no banco de dados e observe que a lista de produtos não é atualizada. Para resolver isso, adicione ao método `GRAVA()` do Managed Bean o código para buscar a lista atualizada no banco de dados após a inserção:

```
public void grava() {
    dao.insere(produto);
    this.produto = new Produto();
    this.produtos = dao.getLista();
}
```

20.4. REMOÇÃO DE PRODUTOS

No formulário visualizamos os produtos cadastrados no sistema e, agora, precisamos adicionar a opção de remoção destes, caso seja necessário.

O primeiro passo é adicionarmos uma coluna na nossa tabela contendo um link para efetuar a remoção. O componente `<H:COMMANDLINK>` gera uma âncora em HTML (link) que funciona como um botão de submissão de formulário.

```
<h:column>
        <h:commandLink action="#{produtoBean.remove}" value="Remover">

        </h:commandLink>
</h:column>
```

Não temos como receber parâmetros nos métodos de ação dos nossos Managed Beans. Então, para saber qual é o produto que queremos remover, o JSF possui um componente para que, ao clicarmos no link que criamos, ele modifique algum atributo do nosso Managed Bean. Essa tag é a `<F:SETPROPERTYACTIONLISTENER>` que recebe dois valores como atributo, um **target** e um **value**.

O `TARGET` é usado para indicar qual a propriedade do Managed Bean que deve ser alterada, e para isso, ela deverá ter um método *setter* que será invocado. Enquanto o atributo `VALUE` indica qual o novo valor que será atribuído.

Sabendo disso, o que precisamos fazer é atribuir o item que estamos iterando na tabela para o atributo value do Managed Bean, da seguinte forma:

```
<h:column>
        <h:commandLink action="#{produtoBean.remove}" value="Remover">
            <f:setPropertyActionListener target="#{produtoBean.produto}"
                value="#{produto}" />
        </h:commandLink>
</h:column>
```

No JSF, todos os botões e links devem estar dentro de um formulário. Para isso, podemos envolver a tabela dentro de um `<H:FORM>`:

```
<h:form >

    <h:dataTable value="#{produtoBean.produtos}" var="produto">

<!--            outras colunas estão aqui -->

        <h:column>
                <h:commandLink action="#{produtoBean.remove}" value="Remover">
                    <f:setPropertyActionListener target="#{produtoBean.produto}"
                        value="#{produto}" />
                </h:commandLink>
        </h:column>

    </h:dataTable>
</h:form>
```

20.5. EXERCÍCIOS: Remoção de produtos

1) Crie o método `REMOVE()` em `PRODUTOBEAN` que fará a exclusão do produto e, após isso, atualizará a listagem:

```
//outros atributos e métodos
public void remove() {
    dao.remove(produto);
    this.produto = new Produto();
    this.produtos = dao.getLista();
}
```

2) Envolva o `DATATABLE` em um novo formulário na página `PRODUTO.XHTML`, pois teremos um `COMMANDLINK` e todos eles devem estar dentro de formulários.

```
    <h:form>
        <h:dataTable value="#{produtoBean.produtos}" var="produto">
<!--            colunas do datatable -->
        </h:dataTable>
    </h:form>
```

3) Crie uma nova coluna no `DATATABLE` contendo um link para fazer a exclusão.

```
<h:dataTable value="#{produtoBean.produtos}" var="produto">

<!--            outras colunas estão aqui -->

    <h:column>
            <h:commandLink action="#{produtoBean.remove}" value="Remover">

            </h:commandLink>
    </h:column>

</h:dataTable>
```

4) Com o `SETPROPERTYACTIONLISTENER`, atribua o produto escolhido:

```
<h:column>
        <h:commandLink action="#{produtoBean.remove}" value="Remover">
            <f:setPropertyActionListener target="#{produtoBean.produto}"
                value="#{produto}" />
        </h:commandLink>
</h:column>
```

5) Reinicie o servidor e acesse a página `PRODUTO.XHTML` no navegador. Exclua alguns produtos.

20.6. EDIÇÃO DE PRODUTOS

Para completarmos a tela de produtos, precisamos alterá-los. Para isso, a abordagem que utilizaremos é a de clicar em um link chamado "Alterar" e os dados do produto deverão aparecer no formulário.

Para isso, criaremos mais um `<H:COMMANDLINK>` dessa vez chamado "Alterar". Ao clicar no link, precisamos pegar o item que ele clicou e exibi-lo no formulário. Repare que o formulário possui um *binding* com o atributo produto. Dessa forma, quaisquer alterações que façamos no atributo produto serão refletidas na tela. Portanto, podemos utilizar a mesma estratégia que usamos para a remoção, que foi utilizar o `SETPROPERTYACTIONLISTENER` para trocar o valor do atributo produto.

```
<h:column>
    <h:commandLink value="Alterar">
        <f:setPropertyActionListener target="#{produtoBean.produto}"
            value="#{produto}" />
    </h:commandLink>
</h:column>
```

Repare, porém, que não temos um método a ser invocado quando o link é clicado (diferentemente da remoção e adição). No caso da edição, apenas queremos carregar os dados do objeto selecionado e o `SETPROPERTYACTIONLISTENER` já é suficiente.

Agora que os dados do produto foram exibidos no formulário e o usuário pode editá-los, quando ele finalizar a edição, ele pode clicar no botão gravar. No entanto, não há como saber se o usuário está adicionando um novo produto ou alterando algum já existente no mesmo método.

Uma forma de fazermos isso é verificar no método grava se o id do produto foi enviado. Caso tenha sido, significa que ele já existia no banco de dados, portanto, devemos alterá-lo, senão, precisamos adicioná-lo no banco de dados.

A primeira alteração que isso nos leva a fazer é no método grava do Managed Bean `PRODUTOBEAN`. Esse método deverá ter agora essa verificação, conforme o código abaixo:

```
public void grava() {
    if(produto.getId() == 0)
        dao.insere(produto);
    else
        dao.altera(produto);
    this.produto = new Produto();
    this.produtos = dao.getLista();
}
```

No entanto, o nosso formulário ainda não envia o id do produto. Precisamos adicionar ainda mais um campo no nosso formulário contendo o `ID`. Podemos passá-lo de forma escondida com um campo `HIDDEN`:

```
<h:form >

<h:inputHidden value="#{produtoBean.produto.id}" />

<!--    outros componentes do h:form -->

</h:form>
```

Mas, ainda não temos como saber visualmente se estamos na verdade adicionando um produto novo ou editando um existente. Uma maneira elegante seria ter um título HTML na página:

```
<h2>
    <h:outputText value="Novo Produto"  />
    <h:outputText value="Editar Produto"     />
</h2>
```

Para mostrar o título dependendo da situação, usaremos o atributo **rendered** que todos os componentes têm para indicar quando renderizá-los. O **rendered** recebe uma expressão booleana e se o valor desta for *false*, o componente não será exibido, e se for *true*, será renderizado para o browser do usuário.

No nosso caso, poderíamos verificar se o id do produto é igual a zero:

```
<h2>
    <h:outputText value="Novo Produto" rendered="#{produtoBean.produto.id==0}" />
    <h:outputText value="Editar Produto" rendered="#{produtoBean.produto.id !=0}" />
</h2>
```

20.7. EXERCÍCIOS: Edição de produtos

1) Altere o método `GRAVA()` em `PRODUTOBEAN` para que ele suporte também a atualização dos dados. Basta utilizar o método altera do DAO caso o objeto possua um ID:

```
public void grava() {
    if(produto.getId() == 0)
        dao.insere(produto);
    else
        dao.altera(produto);
    this.produto = new Produto();
    this.produtos = dao.getLista();
}
```

2) Em `PRODUTO.XHTML` crie o campo id do tipo `HIDDEN`. Coloque-o dentro do form, mas antes do `PANELGRID` para não confundir o layout.

```
<h:form >

<h:inputHidden value="#{produtoBean.produto.id}" />
```

3) Coloque o título do formulário usando o `RENDERED` para mostrar a mensagem correta:

```
<h:body>
        <h2>
            <h:outputText value="Novo Produto" rendered="#{produtoBean.produto.id==0}" />
            <h:outputText value="Editar Produto" rendered="#{produtoBean.produto.id !=0}" />
        </h2>
    <h:form >
```

Referência Bibliográfica:

Apostila Caelum - http://www.caelum.com.br/
http://www.globalcode.com.br/home
http://www.k19.com.br/downloads/apostilas-java
http://www.algaworks.com/
Barnes, David J. e Kolling, Michael. Programação Orientada a Objetos com Java. Pearson
Arnold, Ken, Gosling, James e Holmes, David. A Linguagem de Programação Java. Bookman
Harvey M. Deitel, Paul J. Deitel. Java Como Programar. 6ed. Pearson
Flanagan, David. Guia Essencial do Java. Bookman
Martin, Fowler. UML Essencial.
Livro de Informática – Centro Paula Souza